Boosting

Aula 6

Magno Severino
PADS - Modelos Preditivos
12/06/2021

Na aula passada...

- Árvore de classificação/regressão sofre de variância alta.
- Alternativas:
  - **Bootstrap aggregation**: gera B amostras bootstrap e uma árvore para cada amostra.
  - **Random forest**: similar ao *bagging*, entretanto, considera apenas uma amostra das preditoras a cada divisão das árvores.

Objetivos de aprendizagem

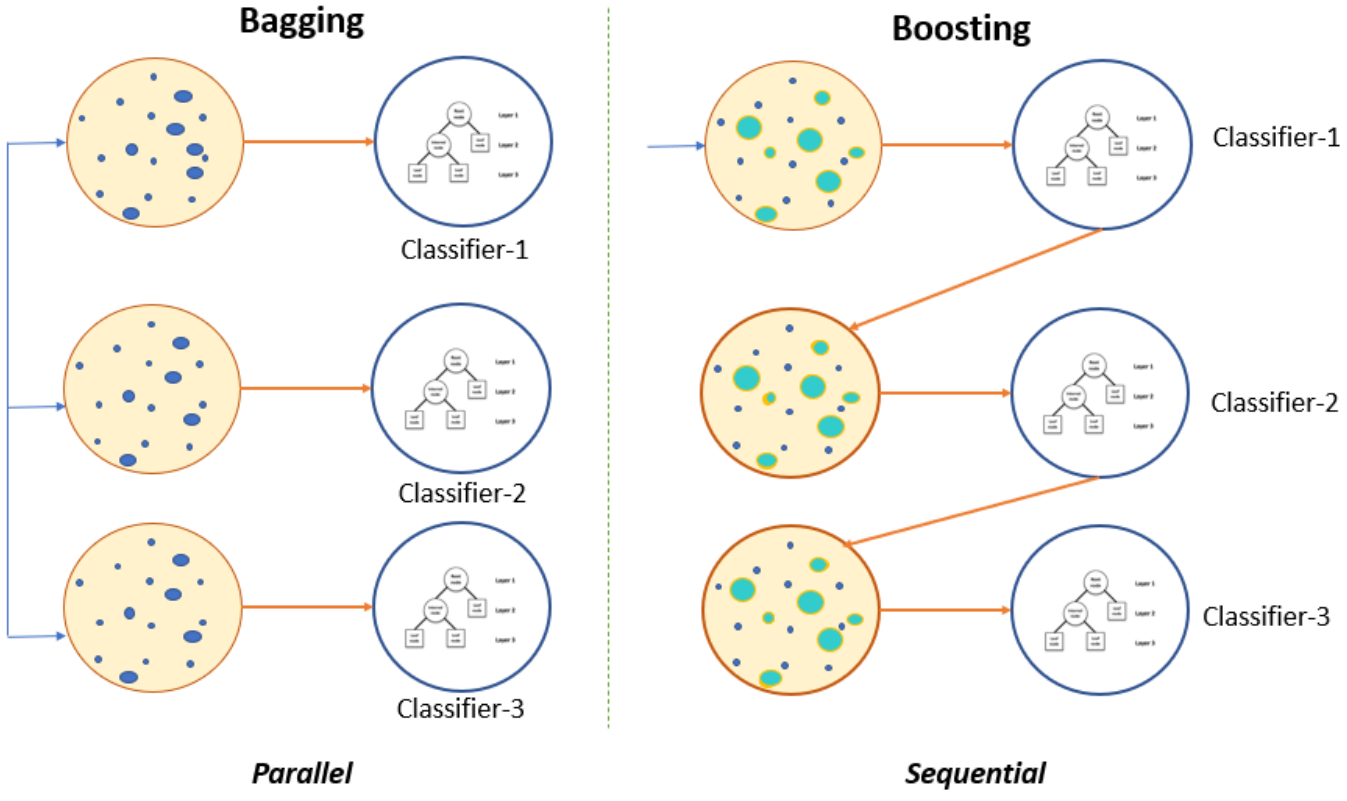
Ao final dessa aula você deverá ser capaz de

- conceituar o método Boosting para classificação e entender as diferenças do Boosting para regressão;
- produzir e interpretar gráficos de dependência parcial.

Boosting

- É um método de *ensemble* ou comitê que parte de classificadores fracos (que são um pouco melhores do que uma predição aleatória) e os agrega, produzindo um classificador de alta performance.
- **Não** se baseia em um mecanismo de reamostragem como o bootstrap.
- Diferentemente da floresta aleatória, no boosting, cada classificador depende do classificador anterior.
- O boosting é um método geral para regressão e classificação, e pode ser utilizado com vários métodos de aprendizagem estatística.
- Vamos trabalhar com boosting para árvores de decisão.

Comparativo



Fonte: Gradient Boosting Trees for Classification: A Beginner's Guide

AdaBoost.M1

- Introduzido por Freund e Schapire em 1997.
- Este trabalho foi reconhecido com o prêmio Gödel em 2003
- O AdaBoost.M1 agrega árvores de classificação geradas pelo algoritmo CART.
- O primeiro é passo é modificar o CART, para que a cada dado de treinamento possa ser atribuído um peso, que será atualizado ao longo do processo de treinamento dos classificadores do ensemble.

AdaBoost.M1

- Considere um problema de classificação com duas classes: $Y \in \{-1, +1\}$ e um dado classificador $\hat{f}(X)$.
- A taxa de erro no conjunto de treinanmento pode ser escrita como

$$\overline{err} = \frac{1}{N} \sum_{i=1}^{N} I\left(y_i \neq \hat{f}(x_i)\right).$$

- Ideia: combinar resultados de M classificadores fracos $\hat{f}_m$ para fazer a previsão a partir de um comitê de classificadores da seguinte forma:

$$\hat{f}(x) = \text{sign}(\sum_{m=1}^{M} \alpha_m \hat{f}_m(x)),$$

em que α_m (para $m = 1, \ldots, M$) são calculados pelo algoritmo e ponderam a contribuição de cada classificador.

Algoritmo AdaBoost.M1

1. Inicialize os pesos fazendo $w_i = \frac{1}{n}$, para $i = 1, \ldots, n$.

2. Para m entre 1 e M:

   2.1 Treine o classificador $\hat{f}_m$ a partir dos dados de treinamento e dos pesos w_i.

   2.2. Calcule $\overline{err}_m = \frac{\sum_{i=1}^{N} w_i I\left(y_i \neq \hat{f}(x_i)\right)}{\sum_{i=1}^{N} w_i}$.

   2.3 Calcule $\alpha_m = \log\left(\frac{1-\overline{err}_m}{\overline{err}_m}\right)$.

   2.4 Atualize os pesos: $w_i = w_i \cdot \exp\{\alpha_m \cdot I(y_i \neq \hat{f}_m(x_i))\}$, $i = 1, \ldots, N$.

3. Classificador agregado: $\hat{f}(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m \hat{f}_m(x)\right)$.

Boosting para regressão

- A cada passo do algoritmo, uma árvore CART é treinada de maneira que os valores dos resíduos atuais sejam as variáveis resposta do problema de regressão.
- Cada árvore de regressão treinada ao longo do algoritmo deve ser baixa.
- Em geral, treina-se um *stump* (toco) com apenas um *split.*
- A intuição é que, ao treinarmos as árvores sucessivamente nos resíduos, estaremos melhorando nosso modelo naqueles dados que não são bem explicados.

Algoritmo Boosting para regressão

Considere os dados de treinamento $\{(x_1, y_1), \ldots, (x_n, y_n)\}$.

1. Defina $\hat{f}(x) = 0$ e $r_i = y_i$ para $i = 1, \ldots, M$.

2. Para b entre 1 e B:

   2.1 Ajuste uma árvore $\hat{f}_b$ com d divisões/*splits* ($d + 1$ nós terminais) utilizando como dados de treinamento $\{(x_1, r_1), \ldots, (x_n, r_n)\}$.

   2.2 Atualize $\hat{f}$ adicionando uma versão encolhida da nova árvore: $\hat{f}(x) = \hat{f}(x) + \lambda \hat{f}_b(x)$.

   2.3 Atualize os resíduos: $r_i = r_i - \lambda \hat{f}_b(x)$.

3. Regressor: $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}_b(x)$

Boosting - hiperparâmetros

- **Número de árvores B**: diferente de bagging e floresta aleatória, boosting pode sobreajustar se B for muito grande, embora o sobreajuste tende a ocorrer vagarosamente (caso ocorra).
- **Parâmetro de encolhimento λ**: esse parâmetro controla a taxa com a qual o método aprende. Os valores típicos são 0.01 e 0.001 e a escolha correta pode depender do problema. Valores muito pequenos de λ podem precisar de um número muito grande de árvores (B) para se alcançar uma boa performance.
- **O número de divisões/split** em cada árvore: esse parâmetro controla a complexidade do método. Normalmente $d = 1$ funciona bem, nesse caso cada árvore é um *stump*, para uma árvore com apenas uma divisão/*split*.

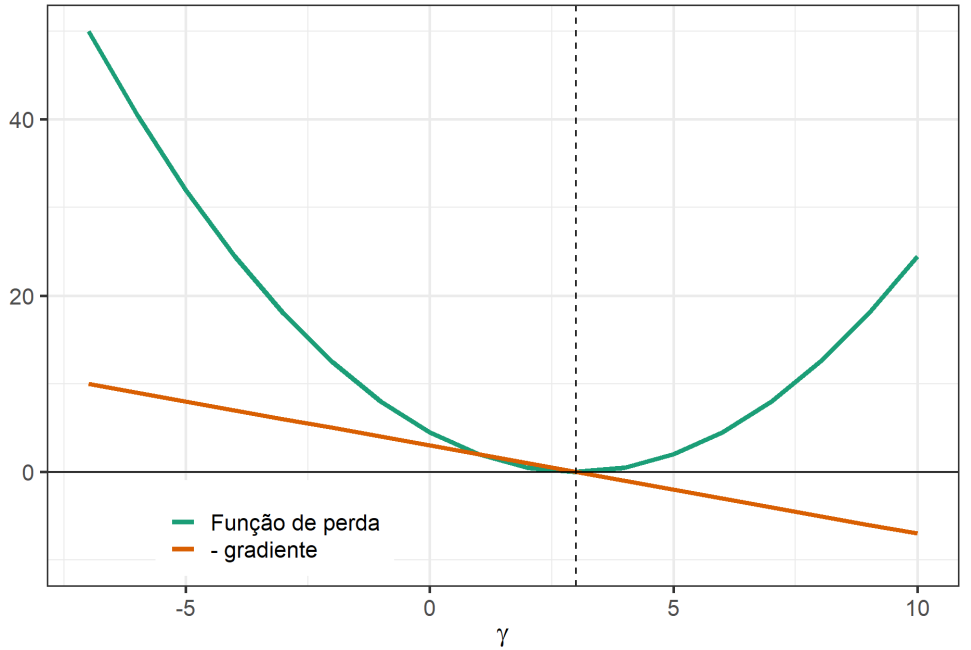
Gradient boosting - função de perda

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \leq \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha\text{th-quantile}\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Tabela retirada do livro The Elements of Statistical Learning

Gradient boosting - função de perda

Com $y_i = 3$ e considere $L(y_i, \gamma)^2$ e $-[\frac{\partial L(y_i,\gamma)}{\partial \gamma}] = (y_i - \gamma)$.



Algoritmo para Gradient Boosting

1. Inicialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.

2. Para m entre 1 e M:

   2.1 Para $i = 1, \ldots, N$, calcule o gradiente residual utilizando $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$.

   2.2 Ajuste uma árvore de regressão utilizando r_{im} para obter os nós terminais/folhas R_{jm}, $j = 1, \ldots, J_m$.

   2.3 Para $j = 1, \ldots, J_m$, calcule γ_{jm} que minimiza $\sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})^2$.

   2.4 Atualize $f_m(x) = f_{m-1}(x) + \lambda \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.

3. Retorne $f(x) = f_M(x)$.

Dados Boston

```
library(MASS)
library(gbm)
library(rsample)

data(Boston)

set.seed(123)

split <- initial_split(Boston, prop = .8)

treinamento <- training(split)

teste <- testing(split)

(fit_bst <- gbm(medv ~ ., distribution = "gaussian",
                n.trees = 5000, interaction.depth = 1,
                shrinkage = 0.1, data = treinamento))
```

```
## gbm(formula = medv ~ ., distribution = "gaussian", data = treinamento,
##     n.trees = 5000, interaction.depth = 1, shrinkage = 0.1)
## A gradient boosted model with gaussian loss function.
## 5000 iterations were performed.
## There were 13 predictors of which 13 had non-zero influence.
```

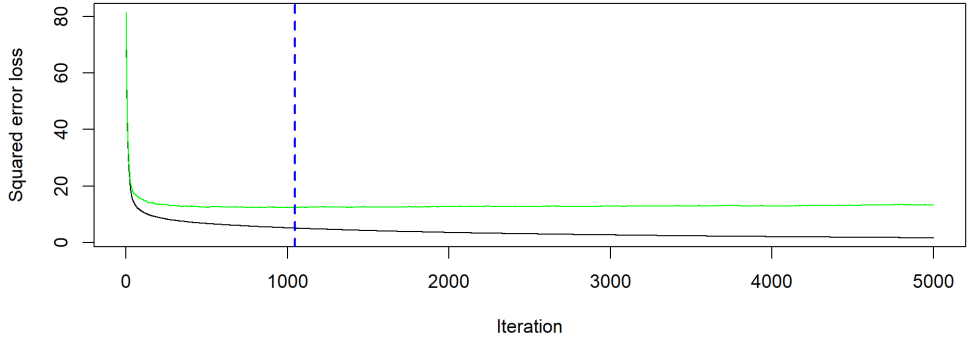
Dados Boston

É possível estimar o número ótimo de iterações do algoritmo

```
boston_cv <- gbm(medv ~ ., data = treinamento, cv.folds = 5,
                 n.trees = 5000, interaction.depth = 1,
                 shrinkage = 0.1)
```

```
## Distribution not specified, assuming gaussian ...
```

```
(ntrees <- gbm.perf(boston_cv, method = "cv"))
```



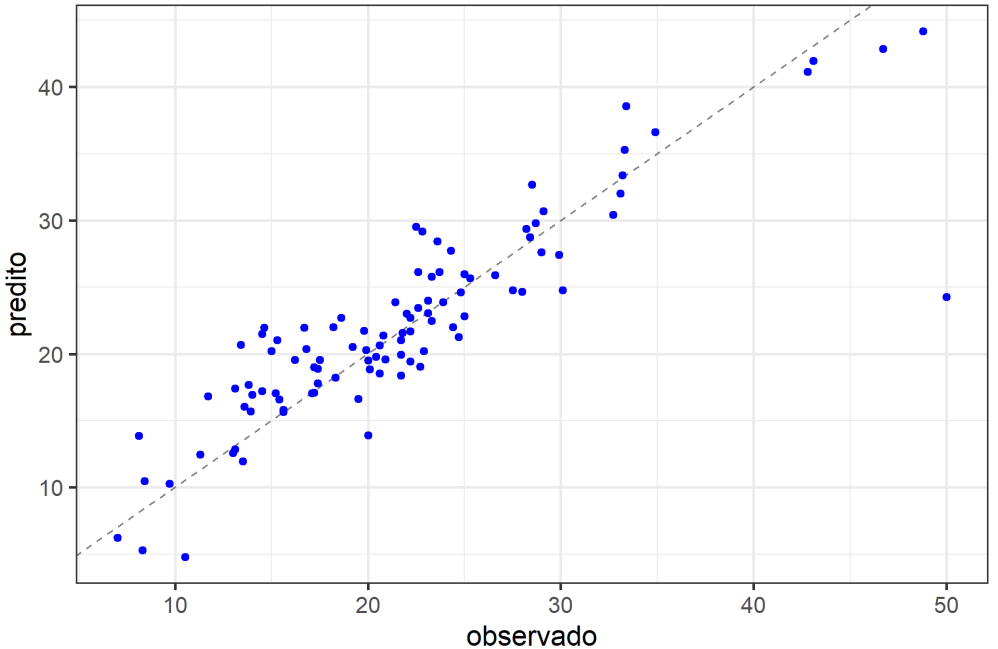

```
## [1] 1046
```

Dados Boston - previsão

```
predito_bst <- predict(fit_bst, newdata = teste,
                       n.trees = ntrees)

(erro_bst <- Metrics::mse(teste$medv, predito_bst))

resultados <- tibble(observado = teste$medv,
                     predito = predito_bst)

ggplot(resultados, aes(observado, predito)) +
  geom_abline(intercept = 0, slope = 1, color = "black",
              alpha = .5, linetype = "dashed") +
  geom_point(color = "blue", size = 2) +
  theme_bw()
```

Dados Boston - previsão

```
## [1] 15.75658
```



Importância de uma variável preditora

Como vimos na última aula, para uma árvore de decisão T, temos a seguinte medida de importância para cada preditora X_l

$$I_l^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 I(v(t) = l).$$

A soma é relativa a todos os $J-1$ nós internos da árvore. Para cada nó t, uma das variáveis $X_v(t)$ é utilizada para particionar a região em duas subregiões e em cada subregião considera-se uma constante para predição. A variável selecionada é a que apresenta o melhor aumento estimado $\hat{i}_t^2$ no *squared error risk* relativo ao ajuste de uma constante para a região inteira. A importância quadrática de X_l é a soma de todos os melhoramentos quadráticos para todos os nós para os quais a variável foi escolhida como variável de divisão/*split*.

Importância relativa de uma variável preditora

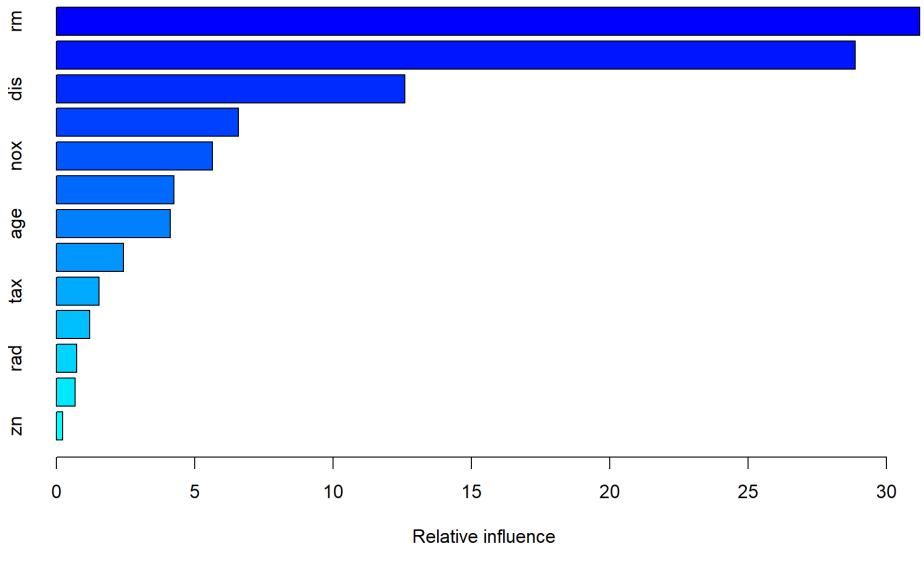
Essa medida de importância é facilmente generalizada para a utilização de diversas árvores da seguinte forma:

$$I_l^2 = \frac{1}{M} \sum_{m=1}^{M} I_l^2(T_m).$$

Como essas medidas são relativas, é comum utilizar o maior valor como 100 e escalar os demais valores de acordo com essa quantidade.

Dados Boston

```
summary(fit_bst)
```




```
##             var    rel.inf
## rm           rm 31.1999499
## lstat     lstat 28.8741960
## dis         dis 12.5847931
## crim       crim  6.5745077
## nox         nox  5.6268380
## black     black  4.2461974
## age         age  4.1177488
## ptratio ptratio  2.4122974
## tax         tax  1.5375097
```

Partial Dependence Plots

Funções de dependência parcial podem ser utilizadas para interpretar os resultados de alguns modelos de aprendizados tidos como *black boxes*. Para uma variável preditora S a função de dependência parcial pode ser estimada utilizando o conjunto de treinamento por

$$\overline{f}_{X_S}(x_S) = \frac{1}{N}\sum_{i=1}^{N} f\left(x_S, x_C^{(i)}\right).$$

Essa função nos diz, para um dado valor da preditora X_S, qual é o seu efeito marginal na predição. Na expressão acima, $x_C^{(i)}$ são os valores no conjunto de treinamento de todas as demais preditoras além de X_S.

Assume-se que, para obter funções de dependência parcial, as preditoras X_S e X_C não são correlacionadas. Se essa suposição for violada, dados que são muito pouco prováveis ou até mesmo impossíveis de serem observados serão considerados.

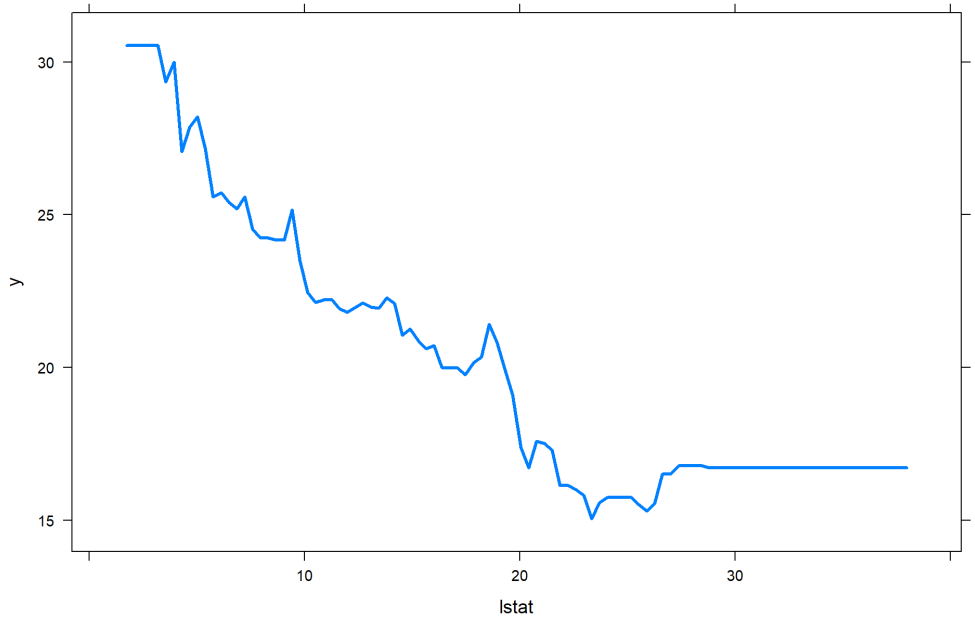
Dados Boston

crim	lstat	rm	zn	nox	age	ptratio	medv
7.52601	19.31	6.417	0.0	0.713	98.3	20.2	13.0
0.03113	10.53	6.014	0.0	0.442	48.5	18.8	17.5
0.03961	8.01	6.037	0.0	0.515	34.5	20.2	21.1
0.33983	9.16	6.108	22.0	0.431	34.9	19.1	24.3
0.06162	12.67	5.898	0.0	0.442	52.3	18.8	17.2
0.09178	9.04	6.416	0.0	0.510	84.1	16.6	23.6
5.09017	17.27	6.297	0.0	0.713	91.8	20.2	16.1
0.13554	13.09	5.594	12.5	0.409	36.8	18.9	17.4
6.39312	24.10	6.162	0.0	0.584	97.4	20.2	13.3
1.15172	18.35	5.701	0.0	0.538	95.0	21.0	13.1
0.55778	16.96	6.335	0.0	0.624	98.2	21.2	18.1
0.04684	8.81	6.417	0.0	0.489	66.1	17.8	22.6
0.04462	7.22	6.619	25.0	0.426	70.4	19.0	23.9
88.97620	17.21	6.968	0.0	0.671	91.9	20.2	10.4
3.16360	14.13	5.759	0.0	0.655	48.2	20.2	19.9

Dados Boston - PDP de `lstat`

`lstat`: lower status of the population (percent).

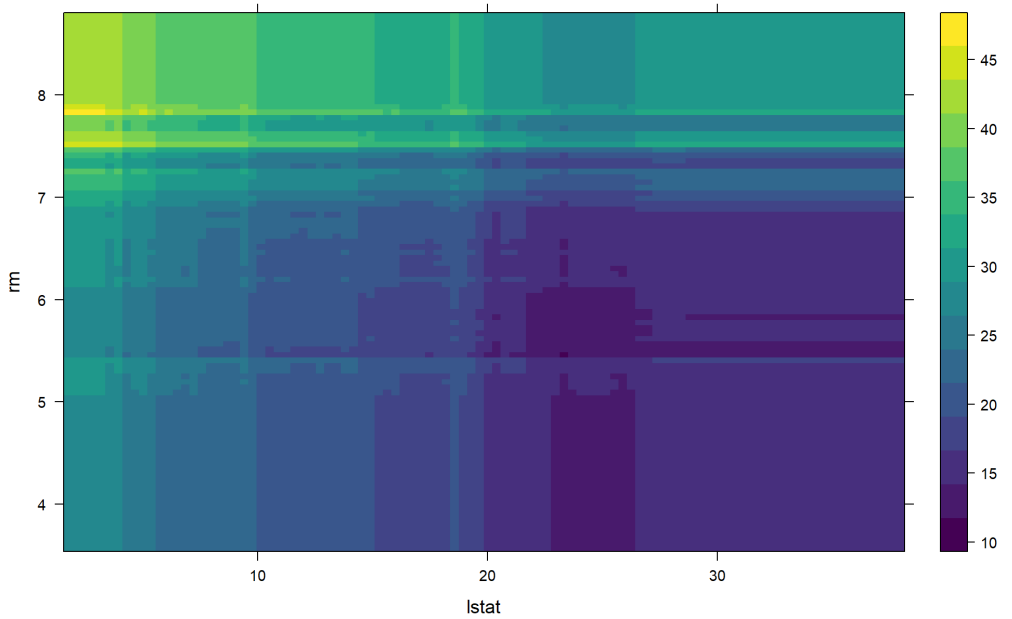
```
plot(fit_bst, i = "lstat", lwd = 3)
```



PDP de `lstat` e `rm`

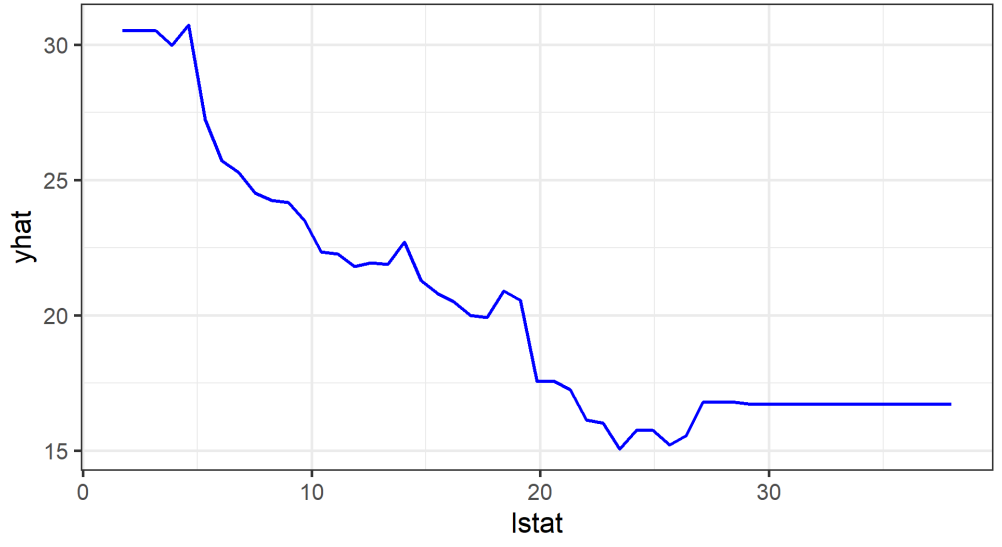
`rm`: average number of rooms per dwelling.

```
plot(fit_bst, i = c("lstat", "rm"))
```



Pacote pdp

```
library(pdp)
pdp::partial(fit_bst, pred.var = "lstat", n.trees = 5000) %>%
  ggplot(aes(lstat, yhat)) +
  geom_line(color = "blue", size = 1)
```

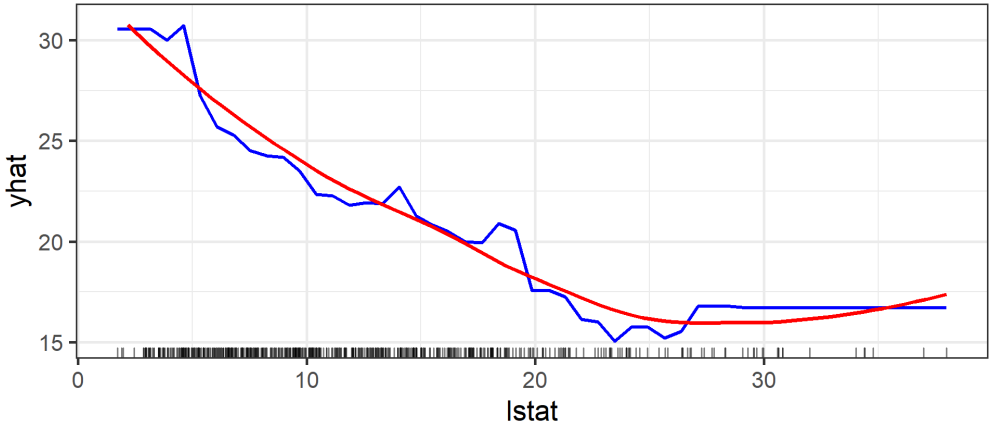


Pacote pdp

```
library(pdp)
pdp::partial(fit_bst, pred.var = "lstat", n.trees = 5000) %>%
  ggplot(aes(lstat, yhat)) +
  geom_line(color = "blue", size = 1) +
  geom_rug(aes(lstat, medv), data = Boston, sides = "b", alpha = 0.5) +
  ylim(15, 31)
```

Pacote pdp

```
library(pdp)
pdp::partial(fit_bst, pred.var = "lstat", n.trees = 5000) %>%
  ggplot(aes(lstat, yhat)) +
  geom_line(color = "blue", size = 1) +
  geom_smooth(color = "red", size = 1.2, se = FALSE) +
  geom_rug(aes(lstat, medv), data = Boston, sides = "b", alpha = 0.5) +
  ylim(15, 31)
```



Implementações do Boosting

Atualmente existem diversas implementações e versões desse método. Além das citadas anteriormente, podemos considerar

- XGBoost;
- LightGBM;
- CatBoost.

A seguir faremos uma aplicação com a biblioteca xgboost.

XGBoost

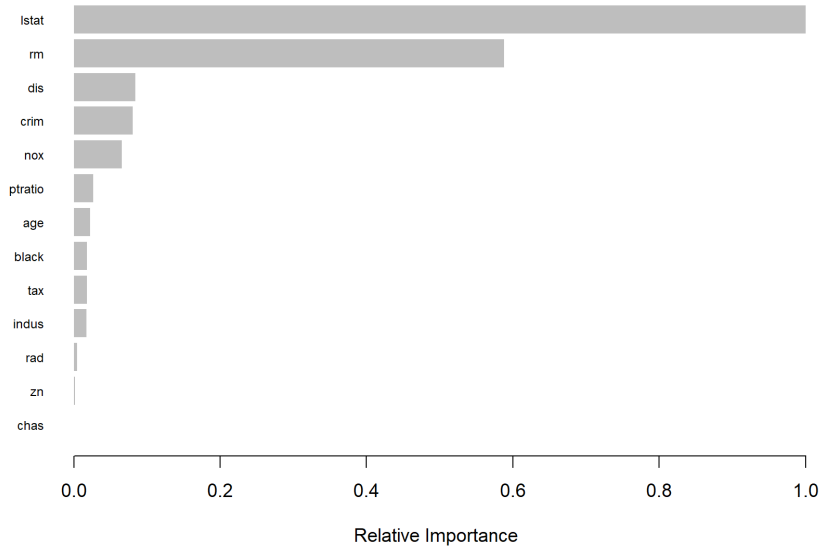
XGBoost (eXtreme Gradient Boosting) é um pacote otimizado para *boosting* com algumas alterações e novas funcionalidades. Com esse pacote é possível realizar processamento em paralelo e atualizar um modelo com base na última iteração caso um novo conjunto de dados esteja disponível.

```
library(xgboost)

d_tr <- xgb.DMatrix(label = treinamento$medv,
                    data = as.matrix(select(treinamento, -medv)))

boston_xgb <- xgboost(data = d_tr, nrounds = 500, max_depth = 4,
                      eta = 0.1, nthread = 3, verbose = FALSE,
                      objective = "reg:squarederror")
```

`eta`: control the learning rate. Used to prevent overfitting by making the boosting process more conservative. Lower value for eta implies larger value for nrounds: low eta value means model more robust to overfitting but slower to compute. Default: 0.3.

XGBoost

```
importancia <- xgb.importance(model = boston_xgb)

xgb.plot.importance(importancia, rel_to_first = TRUE,
                    xlab = "Relative Importance")
```



XGBoost

```
d_test <- xgb.DMatrix(label = teste$medv,
        data = as.matrix(select(teste, -"medv")))

pred_xgb <- predict(boston_xgb, d_test)

Metrics::mse(pred_xgb, teste$medv)
```

```
## [1] 6.845327
```

Resumindo...

- Boosting é um método que pode ser usado tanto para regressão quanto para classificação.
- É uma alternativa para melhorar o poder preditivo de árvore de decisão.
- Gráficos de dependência parcial são uma ferramenta útil para estudar o efeito marginal de uma ou duas variáveis preditoras na variável resposta.

Obrigado!

magnotfs@insper.edu.br